The Embedded I/O Company

TEWS TECHNOLOGIES

TDRV003-SW-42

VxWorks Device Driver

16 (8) Digital Inputs and 16 (8) Digital Outputs

Version 4.0.x

User Manual

Issue 4.0.0

July 2013

TEWS TECHNOLOGIES GmbH
Am Bahnhof 7 25469 Halstenbek, Germany
Phone: +49 (0) 4101 4058 0 Fax: +49 (0) 4101 4058 19
e-mail: info@tews.com www.tews.com

TDRV003-SW-42

VxWorks Device Driver

16 (8) Digital Inputs and 16 (8) Digital Outputs

Supported Modules:
TPMC670
TPMC671

This document contains information, which is proprietary to TEWS TECHNOLOGIES GmbH. Any reproduction without written permission is forbidden.

TEWS TECHNOLOGIES GmbH has made any effort to ensure that this manual is accurate and complete. However TEWS TECHNOLOGIES GmbH reserves the right to change the product described in this document at any time without notice.

TEWS TECHNOLOGIES GmbH is not liable for any damage arising out of the application or use of the device described herein.

©2005-2013 by TEWS TECHNOLOGIES GmbH

Issue	Description	Date
1.0.0	First Issue	October 14, 2005
1.0.1	New Address TEWS LLC, general revision	October 21, 2008
2.0.0	API documentation added	September 6, 2010
2.0.1	General Revision	January 25, 2011
3.0.0	VxWorks 6.9 support, 64-bit support	February 07, 2012
4.0.0	API modified, Basic I/O functions removed	July 04, 2013

Table of Contents

1 INTRODUCTION 4

2 INSTALLATION 5

2.1 Legacy vs. VxBus Driver 6

2.2 VxBus Driver Installation 6

2.2.1 Direct BSP Builds 8

2.3 Legacy Driver Installation 8

2.3.1 Include Device Driver in VxWorks Projects 8

2.3.2 Special Installation for Intel x86 based Targets 8

2.3.3 BSP dependent Adjustments 9

2.3.4 System Resource Requirement 10

2.4 Special Configurations 10

3 API DOCUMENTATION 11

3.1 General Functions 11

3.1.1 tdrv003Open 11

3.1.2 tdrv003Close 13

3.2 Device Access Functions 15

3.2.1 tdrv003InputRead 15

3.2.2 tdrv003OutputWrite 17

3.2.3 tdrv003OutputRead 19

3.2.4 tdrv003OutputWriteMask 21

3.2.5 tdrv003OutputSetBits 23

3.2.6 tdrv003OutputClearBits 25

3.2.7 tdrv003EventWait 27

3.2.8 tdrv003DebouncerEnable 29

3.2.9 tdrv003DebouncerDisable 31

3.2.10 tdrv003WatchdogEnable 33

3.2.11 tdrv003WatchdogDisable 35

3.2.12 tdrv003WatchdogReset 37

4 LEGACY I/O SYSTEM FUNCTIONS 39

4.1 tdrv003PciInit 39

5 DEBUGGING AND DIAGNOSTIC 40

1 Introduction

The TDRV003-SW-42 VxWorks device driver software allows the operation of the modules supported by TDRV003 conforming to the VxWorks I/O system specification.

The TDRV003-SW-42 release contains independent driver sources for the old legacy (pre-VxBus) and the new VxBus-enabled driver model. The VxBus-enabled driver is recommended for new developments with later VxWorks 6.x releases and mandatory for VxWorks SMP systems.

Both drivers, legacy and VxBus, share the same application programming interface (API) and device-independent basic I/O interface with open(), close() and ioctl() functions. The basic I/O interface is only for backward compatibility with existing applications and should not be used for new developments.

Both drivers invoke a mutual exclusion and binary semaphore mechanism to prevent simultaneous requests by multiple tasks from interfering with each other.

The TDRV003-SW-42 device driver supports the following features:

- read the actual input value
- write a new output value
- wait for selectable input events (match, high-, low-, any transition on the input line(s))
- enable and disable the output watchdog
- acknowledge watchdog errors
- configure, enable and disable input debouncing

The TDRV003-SW-42 supports the modules listed below:

TPMC670	16(8) Digital Input (24V) 16(8) Digital Output (24V, 0.5A) (50 pin connector)	PMC
TPMC671	16 Digital Input (24V) 16 Digital Output (24V, 0.5A) (64 pin connector)	PMC

In this document all supported modules and devices will be called TDRV003. Specials for certain devices will be advised.

To get more information about the features and use of TDRV003 devices, it is recommended to read the manuals listed below.

TPMC670/TPMC671 User Manual
TPMC670/TPMC671 Engineering Manual

2 Installation

The following files are located on the distribution media:

Directory path 'TDRV003-SW-42':

TDRV003-SW-42-4.0.0.pdf	PDF copy of this manual
TDRV003-SW-42-VXBUS.zip	Zip compressed archive with VxBus driver sources
TDRV003-SW-42-LEGACY.zip	Zip compressed archive with legacy driver sources
ChangeLog.txt	Release history
Release.txt	Release information

The archive TDRV003-SW-42-VXBUS.zip contains the following files and directories:

Directory path './tews/tdrv003':

tdrv003drv.c	TDRV003 device driver source
tdrv003def.h	TDRV003 driver include file
tdrv003.h	TDRV003 include file for driver and application
tdrv003api.h	TDRV003 API include file
tdrv003api.c	TDRV003 API file
Makefile	Driver Makefile
40tdrv003.cdf	Component description file for VxWorks development tools
tdrv003.dc	Configuration stub file for direct BSP builds
tdrv003.dr	Configuration stub file for direct BSP builds
include/tvxbHal.h	Hardware dependent interface functions and definitions
apps/tdrv003exa.c	Example application

The archive TDRV003-SW-42-LEGACY.zip contains the following files and directories:

Directory path './tdrv003':

tdrv003drv.c	TDRV003 device driver source
tdrv003def.h	TDRV003 driver include file
tdrv003.h	TDRV003 include file for driver and application
tdrv003pci.c	TDRV003 device driver source for x86 based systems
tdrv003api.h	TDRV003 API include file
tdrv003api.c	TDRV003 API file
tdrv003exa.c	Example application
include/tdhal.h	Hardware dependent interface functions and definitions

2.1 Legacy vs. VxBus Driver

In later VxWorks 6.x releases, the old VxWorks 5.x legacy device driver model was replaced by VxBus-enabled device drivers. Legacy device drivers are tightly coupled with the BSP and the board hardware. The VxBus infrastructure hides all BSP and hardware differences under a well defined interface, which improves the portability and reduces the configuration effort. A further advantage is the improved performance of API calls by using the method interface and bypassing the VxWorks basic I/O interface.

VxBus-enabled device drivers are the preferred driver interface for new developments.

The checklist below will help you to make a decision which driver model is suitable and possible for your application:

Legacy Driver	VxBus Driver
▪ VxWorks 5.x releases ▪ VxWorks 6.5 and earlier releases ▪ VxWorks 6.x releases without VxBus PCI bus support	▪ VxWorks 6.6 and later releases with VxBus PCI bus ▪ SMP systems (only the VxBus driver is SMP safe) ▪ 64-bit systems (only the VxBus driver is 64-bit compatible)

TEWS TECHNOLOGIES recommends not using the VxBus Driver before VxWorks release 6.6. In previous releases required header files are missing and the support for 3rd-party drivers may not be available.

2.2 VxBus Driver Installation

Because Wind River doesn't provide a standard installation method for 3rd party VxBus device drivers, the installation procedure needs to be done manually.

In order to perform a manual installation, extract all files from the archive TDRV003-SW-42-VXBUS.zip to the typical 3rd party directory *installDir/vxworks-6.x/target/3rdparty* (whereas *installDir* must be substituted by the VxWorks installation directory).

After successful installation, the TDRV003 device driver is located in the vendor and driver-specific directory *installDir/vxworks-6.x/target/3rdparty/tews/tdrv003.*

At this point the TDRV003 driver is not configurable and cannot be included with the kernel configuration tool in a Wind River Workbench project. To make the driver configurable, the driver library for the desired processor (CPU) and build tool (TOOL) must be built in the following way:

(1) Open a VxWorks development shell (e.g. C:\WindRiver\wrenv.exe -p vxworks-6.7)

(2) Change into the driver installation directory
installDir/vxworks-6.x/target/3rdparty/tews/tdrv003

(3) Invoke the build command for the required processor and build tool with optional VXBUILD argument
make CPU=cpuName TOOL=tool [VXBUILD=xxx]

For Windows hosts this may look like this:

```
C:> cd \WindRiver\vxworks-6.7\target\3rdparty\tews\tdrv003
C:> make CPU=PENTIUM4 TOOL=diab
```

To compile SMP-enabled libraries, the argument VXBUILD=SMP must be added to the command line

```
C:> make CPU=PENTIUM4 TOOL=diab VXBUILD=SMP
```

To build 64-bit libraries, the argument VXBUILD=LP64 must be added to the command line

```
> make TOOL=gnu CPU=CORE VXBUILD=LP64
```

For 64-bit SMP-enabled libraries a build command may look like this

```
> make TOOL=gnu CPU=CORE VXBUILD="LP64 SMP"
```

To integrate the TDRV003 driver with the VxWorks development tools (Workbench), the component configuration file *40tdrv003.cdf* must be copied to the directory *installDir/vxworks-6.x/target/config/comps/VxWorks*.

```
C:> cd \WindRiver\vxworks-6.7\target\3rdparty\tews\tdrv003
C:> copy 40tdrv003.cdf \Windriver\vxworks-6.7\target\config\comps\vxWorks
```

In VxWorks 6.7 and newer releases the kernel configuration tool scans the CDF file automatically and updates the *CxrCat.txt* cache file to provide component parameter information for the kernel configuration tool as long as the timestamp of the copied CDF file is newer than the one of the *CxrCat.txt*. If your copy command preserves the timestamp, force to update the timestamp by a utility, such as *touch*.

In earlier VxWorks releases the CxrCat.txt file may not be updated automatically. In this case, remove or rename the original *CxrCat.txt* file and invoke the make command to force recreation of this file.

```
C:> cd \Windriver\vxworks-6.7\target\config\comps\vxWorks
C:> del CxrCat.txt
C:> make
```

After successful completion of all steps above and restart of the Wind River Workbench, the TDRV003 driver can be included in VxWorks projects by selecting the *"TEWS TDRV003 Driver"* component in the *"hardware (default) - Device Drivers"* folder with the kernel configuration tool.

2.2.1 Direct BSP Builds

In development scenarios with the direct BSP build method without using the Workbench or the vxprj command-line utility, the TDRV003 configuration stub files must be copied to the directory *installDir/vxworks-6.x/target/config/comps/src/hwif.* Afterwards, the *vxbUsrCmdLine.c* file must be updated by invoking the appropriate make command.

```
C:> cd \WindRiver\vxworks-6.7\target\3rdparty\tews\tdrv003
C:> copy tdrv003.dc \Windriver\vxworks-6.7\target\config\comps\src\hwif
C:> copy tdrv003.dr \Windriver\vxworks-6.7\target\config\comps\src\hwif

C:> cd \Windriver\vxworks-6.7\target\config\comps\src\hwif
C:> make vxbUsrCmdLine.c
```

2.3 Legacy Driver Installation

2.3.1 Include Device Driver in VxWorks Projects

For including the TDRV003-SW-42 device driver into a VxWorks project (e.g. Tornado IDE or Workbench) follow the steps below:

(1) Extract all files from the archive TDRV003-SW-42-LEGACY.zip to your project directory.

(2) Add the device drivers C-files to your project.
Make a right click to your project in the 'Workspace' window and use the 'Add Files ...' topic.
A file select box appears, and the driver files in the tdrv003 directory can be selected.

(3) Now the driver is included in the project and will be built with the project.

For a more detailed description of the project facility, please refer to your VxWorks User's Guide (e.g. Tornado, Workbench, etc.)

2.3.2 Special Installation for Intel x86 based Targets

The TDRV003 device driver is fully adapted for Intel x86 based targets. This is done by conditional compilation directives inside the source code and controlled by the VxWorks global defined macro **CPU_FAMILY**. If the content of this macro is equal to *I80X86,* special Intel x86 conforming code and function calls will be included.

The second problem for Intel x86 based platforms can't be solved by conditional compilation directives. Due to the fact that some Intel x86 BSP's doesn't map PCI memory spaces of devices which are not used by the BSP, the required device memory spaces can't be accessed.

To solve this problem, a MMU mapping entry has to be added for the required TDRV003 PCI memory spaces prior the MMU initialization (*usrMmuInit()*) is done.

The C source file **tdrv003pci.c** contains the function *tdrv003PciInit().* This routine finds out all TDRV003 devices and adds MMU mapping entries for all used PCI memory spaces. Please insert a call to this function after the PCI initialization is done and prior to MMU initialization (*usrMmuInit()*).

The right place to call the function *tdrv003PciInit()* is at the end of the function *sysHwInit()* in **sysLib.c** (it can be opened from the project *Files* window):

```
tdrv003PciInit();
```

Be sure that the function is called prior to MMU initialization or otherwise the TDRV003 PCI spaces remain unmapped and an access fault occurs during driver initialization.

Modifying the sysLib.c file will change the sysLib.c in the BSP path. Remember this for future projects and recompilations.

2.3.3 BSP dependent Adjustments

The driver includes a file called *include/tdhal.h* which contains functions and definitions for BSP adaptation. It may be necessary to modify them for BSP specific settings. Most settings can be made automatically by conditional compilation set by the BSP header files, but some settings must be configured manually. There are two ways of modification, first you can change the *include/tdhal.h* and define the corresponding definition and its value, or you can do it using the command line option *–D*.

There are 3 offset definitions (*USERDEFINED_MEM_OFFSET*, *USERDEFINED_IO_OFFSET*, and *USERDEFINED_LEV2VEC*) that must be configured if a corresponding warning message appears during compilation. These definitions always need values. Definition values can be assigned by command line option *-D<definition>=<value>*.

Definition	Description
USERDEFINED_MEM_OFFSET	The value of this definition must be set to the offset between CPU-Bus and PCI-Bus Address for PCI memory space access
USERDEFINED_IO_OFFSET	The value of this definition must be set to the offset between CPU-Bus and PCI-Bus Address for PCI I/O space access
USERDEFINED_LEV2VEC	The value of this definition must be set to the difference of the interrupt vector (used to connect the ISR) and the interrupt level (stored to the PCI header)

Another definition allows a simple adaptation for BSPs that utilize a *pciIntConnect()* function to connect shared (PCI) interrupts. If this function is defined in the used BSP, the definition of *USERDEFINED_SEL_PCIINTCONNECT* should be enabled. The definition by command line option is made by *-D<definition>.*

Please refer to the BSP documentation and header files to get information about the interrupt connection function and the required offset values.

2.3.4 System Resource Requirement

The table gives an overview over the system resources that will be needed by the driver.

Resource	Driver requirement	Devices requirement
Memory	~52 Bytes + (<max jobs> * ~12 Bytes)	~52 Bytes
Stack	< 200 Byte	---
Semaphores	0	<max jobs>

<max jobs> = *TDRV003_NUMJOBS*

Memory and Stack usage may differ from system to system, depending on the used compiler and its setup.

The following formula shows the way to calculate the common requirements of the driver and devices.

*<total requirement> = <driver requirement> + (<number of devices> * <device requirement>)*

The maximum usage of some resources is limited by adjustable parameters. If the application and driver exceed these limits, increase the according values in your project.

2.4 Special Configurations

The value *TDRV003_NUMJOBS* (defined in *tdrv003def.h*) specifies the maximum number of jobs that can be handled at the same time for a device. System resources will be allocated while driver start to prevent extra time during normal application operation. This value may be changed to increase the maximum number of jobs or decreased to save system resources.

The driver must be built again after changing *TDRV003_NUMJOBS*.

3 API Documentation

3.1 General Functions

3.1.1 tdrv003Open

NAME

tdrv003Open – Opens a Device

SYNOPSIS

```
TDRV003_HANDLE tdrv003Open
(
      char          *deviceName
);
```

DESCRIPTION

Before I/O can be performed to a device, a file descriptor must be opened by a call to this function.

PARAMETERS

deviceName

This parameter points to a null-terminated string that specifies the name of the device. The first TDRV003 device is named “/tdrv003/0”, the second device is named “/tdrv003/1” and so on.

EXAMPLE

```
#include “tdrv003api.h”

TDRV003_HANDLE hdl;

/*
** open file descriptor to device
*/
hdl = tdrv003Open(“/tdrv003/0”);
if (hdl == NULL)
{
     /* handle open error */
}
```

RETURNS

A device descriptor pointer, or NULL if the function fails. An error code will be stored in *errno*.

ERROR CODES

All error codes are standard error codes set by the I/O system.

3.1.2 tdrv003Close

NAME

tdrv003Close – Closes a Device

SYNOPSIS

```
TDRV003_STATUS tdrv003Close
(
      TDRV003_HANDLE              hdl
);
```

DESCRIPTION

This function closes previously opened devices.

PARAMETERS

hdl

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

EXAMPLE

```
#include “tdrv003api.h”

TDRV003_HANDLE hdl;
TDRV003_STATUS result;

/*
** close file descriptor to device
*/
result = tdrv003Close( hdl );

if (result != TDRV003_OK)
{
     /* handle close error */
}
```

RETURNS

On success TDRV003_OK, or an appropriate error code.

ERROR CODES

Error Code	Description
TDRV003_ERR_INVALID_HANDLE	The specified TDRV003_HANDLE is invalid.

3.2 Device Access Functions

3.2.1 tdrv003InputRead

NAME

tdrv003InputRead – read state of input lines

SYNOPSIS

```
TDRV003_STATUS tdrv003InputRead
(
    TDRV003_HANDLE          hdl,
    unsigned short          *pInputBuf
);
```

DESCRIPTION

This function reads the current state of the input lines.

PARAMETERS

hdl

This argument specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

pInputBuf

This argument points to a buffer where the value will be returned.

EXAMPLE

```
#include “tdrv003api.h”

TDRV003_HANDLE   hdl;
TDRV003_STATUS   result;
unsigned short   data;


result = tdrv003InputRead(hdl, &data);

if (result != TDRV003_OK)
{
     /* handle error */
}
```

RETURNS

On success, TDRV003_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV003_ERR_INVALID_HANDLE	The specified TDRV003_HANDLE is invalid.

3.2.2 tdrv003OutputWrite

NAME

tdrv003OutputWrite – write a new value to the output port

SYNOPSIS

```
TDRV003_STATUS tdrv003OutputWrite
(
      TDRV003_HANDLE      hdl,
      unsigned short      outputValue
);
```

DESCRIPTION

This function writes a new value to the output port.

PARAMETERS

hdl

This argument specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

outputValue

This argument specifies the new output value. Bit 0 specifies the new state of output line 1; bit 1 specifies the new state for output line 2 and so on.

EXAMPLE

```
#include “tdrv003api.h”

TDRV003_HANDLE   hdl;
TDRV003_STATUS   result;

/* set OUTPUT1 and OUTPUT16 and clear all other */
result = tdrv003OutputWrite(hdl, 0x8001);

if (result != TDRV003_OK)
{
     /* handle error */
}
```

RETURNS

On success, TDRV003_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV003_ERR_INVALID_HANDLE	The specified TDRV003_HANDLE is invalid.
TDRV003_ERR_IO	The output register is locked by a watchdog failure. Execute the function tdrv003WatchdogReset to reset the watchdog error.

3.2.3 tdrv003OutputRead

NAME

tdrv003OutputRead – read current state of output lines

SYNOPSIS

```
TDRV003_STATUS tdrv003OutputRead
(
    TDRV003_HANDLE      hdl,
    unsigned short      *pOutputBuf
);
```

DESCRIPTION

This function reads the current state of the output lines.

PARAMETERS

hdl

This argument specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

pOutputBuf

This argument points to a buffer where the value will be returned.

EXAMPLE

```
#include “tdrv003api.h”

TDRV003_HANDLE   hdl;
TDRV003_STATUS   result;
unsigned short   data;


result = tdrv003OutputRead(hdl, &data);

if (result != TDRV003_OK)
{
     /* handle error */
}
```

RETURNS

On success, TDRV003_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV003_ERR_INVALID_HANDLE	The specified TDRV003_HANDLE is invalid.
TDRV003_ERR_IO	The output register is locked by a watchdog failure. Execute the function tdrv003WatchdogReset to reset the watchdog error.

3.2.4 tdrv003OutputWriteMask

NAME

tdrv003OutputWriteMask – writes a masked value to the output port

SYNOPSIS

```
TDRV003_STATUS tdrv003OutputWriteMask
(
    TDRV003_HANDLE          hdl,
    unsigned short          outputValue,
    unsigned short          mask
);
```

DESCRIPTION

This control function writes a masked value to the output port. Only those bits in value for which the corresponding bit position in the mask is set to 1 will be changed in the output port.

PARAMETERS

hdl

This argument specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

outputValue

This argument specifies the masked value for the output port. Bit 0 corresponds to the first output line; bit 1 corresponds to the second output line and so on.

mask

This argument specifies the mask for relevant bits. Bit 0 corresponds to the first output line; bit 1 corresponds to the second output line and so on.
Only those bits in *outputValue* for which the corresponding bit position in this mask is set to 1 will be changed in the output port.

EXAMPLE

```
#include “tdrv003api.h”

TDRV003_HANDLE  hdl;
TDRV003_STATUS  result;

/* clear OUTPUT1 and set OUTPUT16 and leave all other lines unchanged */
result = tdrv003OutputWriteMask(hdl, 0x8000, 0x8001);

if (result != TDRV003_OK)
{
     /* handle error */
}
```

RETURNS

On success, TDRV003_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV003_ERR_INVALID_HANDLE	The specified TDRV003_HANDLE is invalid.
TDRV003_ERR_IO	The output register is locked by a watchdog failure. Execute the function tdrv003WatchdogReset to reset the watchdog error.

3.2.5 tdrv003OutputSetBits

NAME

tdrv003OutputSetBits – set specific output lines

SYNOPSIS

```
TDRV003_STATUS tdrv003OutputSetBits
(
      TDRV003_HANDLE          hdl,
      unsigned short          outputBits
);
```

DESCRIPTION

This function sets specific bits in the output port to active state (1).

PARAMETERS

hdl

This argument specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

outputBits

This argument specifies a mask of relevant bits to set. Bit 0 corresponds to the first output line; bit 1 corresponds to the second output line and so on. Bits which are set (1) in this mask will be set in the corresponding bits in the output port.

EXAMPLE

```
#include “tdrv003api.h”

TDRV003_HANDLE   hdl;
TDRV003_STATUS   result;

/* set OUTPUT1 and OUTPUT16 to active state */
result = tdrv003OutputSetBits(hdl, (1<<15) | (1<<0));

if (result != TDRV003_OK)
{
     /* handle error */
}
```

RETURNS

On success, TDRV003_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV003_ERR_INVALID_HANDLE	The specified TDRV003_HANDLE is invalid.
TDRV003_ERR_IO	The output register is locked by a watchdog failure. Execute the function tdrv003WatchdogReset to reset the watchdog error.

3.2.6 tdrv003OutputClearBits

NAME

tdrv003OutputClearBits – clear specific output lines

SYNOPSIS

```
TDRV003_STATUS tdrv003OutputClearBits
(
      TDRV003_HANDLE          hdl,
      unsigned short          outputBits
);
```

DESCRIPTION

This function clears specific bits in the output port to inactive state (0).

PARAMETERS

hdl

This argument specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

outputBits

This argument specifies a mask of relevant bits to clear. Bit 0 corresponds to the first output line; bit 1 corresponds to the second output line and so on. Bits which are set (1) in this mask will be cleared in the corresponding bits in the output port.

EXAMPLE

```
#include “tdrv003api.h”

TDRV003_HANDLE   hdl;
TDRV003_STATUS   result;

/* clear OUTPUT1 and OUTPUT16 */
result = tdrv003OutputSetBits(hdl, (1<<15) | (1<<0));

if (result != TDRV003_OK)
{
     /* handle error */
}
```

RETURNS

On success, TDRV003_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV003_ERR_INVALID_HANDLE	The specified TDRV003_HANDLE is invalid.
TDRV003_ERR_IO	The output register is locked by a watchdog failure. Execute the function tdrv003WatchdogReset to reset the watchdog error.

3.2.7 tdrv003EventWait

NAME

tdrv003EventWait – wait for a specific input event

SYNOPSIS

```
TDRV003_STATUS tdrv003EventWait
(
    TDRV003_HANDLE      hdl,
    unsigned short      mode,
    unsigned short      mask,
    long                timeout
);
```

DESCRIPTION

This function waits for an event (rising or falling edge or both) at the specified input lines. The function is blocked until at least one of the specified events or a timeout occurs.

PARAMETERS

hdl

This argument specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

mode

This argument specifies the kind of event to wait for. The following event types are defined in tdrv003api.h:

Event Type	Description
TDRV003_HIGH_TR	Wait for a low to high transition on a specified input line.
TDRV003_LOW_TR	Wait for a high to low transition on a specified input line.
TDRV003_ANY_TR	Wait for any transition on a specified input line.

mask

This argument specifies a bit mask of input lines to wait for the specified edge event. Bit 0 corresponds to the first input line; bit 1 corresponds to the second input line and so on.
Only one bit shall be set in the mask, otherwise the occurred event cannot be determined exactly. If more than one bit is set in the mask the function is completed the moment a relevant transition at least at one specified bit position occurs.

timeout

Specifies the amount of time (in milliseconds) the caller is willing to wait for the specified event to occur. A value of 0 means wait indefinitely.

EXAMPLE

```
#include “tdrv003api.h”

TDRV003_HANDLE   hdl;
TDRV003_STATUS   result;

/* wait at least 5 s for any edge at INPUT16 */
result = tdrv003EventWait (hdl, TDRV003_ANY_TR, 1<<15, 5000);

if (result != TDRV003_OK)
{
     /* handle error */
}
```

RETURNS

On success, TDRV003_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV003_ERR_INVALID_HANDLE	The specified TDRV003_HANDLE is invalid.
TDRV003_ERR_BUSY	No more free entries in the drivers queue to handle concurrent event-controlled read requests. Increase the queue size (see also 2.4).
TDRV003_ERR_TIMEOUT	The requested event does not occur within the specified time (timeout).

3.2.8 tdrv003DebouncerEnable

NAME

tdrv003DebouncerEnable – configure and enable debouncer circuit

SYNOPSIS

```
TDRV003_STATUS tdrv003DebouncerEnable
(
    TDRV003_HANDLE      hdl,
    unsigned short      debounceTimer
);
```

DESCRIPTION

This function configures and enables the input debouncer circuit.

PARAMETERS

hdl

This argument specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

debounceTimer

This argument specifies the debouncer time. The debouncer time is specified in approx. 7µs steps. Please refer to the corresponding Hardware User Manual for a calculation formula and a table of values.

EXAMPLE

```
#include “tdrv003api.h”

TDRV003_HANDLE  hdl;
TDRV003_STATUS  result;

/* Enable the debouncer with a debounce time of 1ms */
result = tdrv003DebouncerEnable(hdl, 147);

if (result != TDRV003_OK)
{
     /* handle error */
}
```

RETURNS

On success, TDRV003_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV003_ERR_INVALID_HANDLE	The specified TDRV003_HANDLE is invalid.

3.2.9 tdrv003DebouncerDisable

NAME

tdrv003DebouncerDisable – disable debouncer circuit

SYNOPSIS

```
TDRV003_STATUS tdrv003DebouncerDisable
(
    TDRV003_HANDLE          hdl
);
```

DESCRIPTION

This function disables the input debouncer circuit.

PARAMETERS

hdl

This argument specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

EXAMPLE

```
#include “tdrv003api.h”

TDRV003_HANDLE   hdl;
TDRV003_STATUS   result;


/* Disable the debouncer circuit */
result = tdrv003DebouncerDisable(hdl);

if (result != TDRV003_OK)
{
    /* handle error */
}
```

RETURNS

On success, TDRV003_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV003_ERR_INVALID_HANDLE	The specified TDRV003_HANDLE is invalid.

3.2.10 tdrv003WatchdogEnable

NAME

tdrv003WatchdogEnable – enable output watchdog

SYNOPSIS

```
TDRV003_STATUS tdrv003WatchdogEnable
(
     TDRV003_HANDLE              hdl
);
```

DESCRIPTION

This function enables the watchdog timer for the output lines. The watchdog function is activated after the next write operation to the device. Please remember that if the watchdog is enabled and no write access occurs within 120 ms, all outputs go into the inactive (0) state. To unlock the output register and leave the inactive state the function *tdrv003WatchdogReset* must be executed.

PARAMETERS

hdl

This argument specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

EXAMPLE

```
#include “tdrv003api.h”

TDRV003_HANDLE   hdl;
TDRV003_STATUS   result;


/* Enable output watchdog */
result = tdrv003WatchdogEnable(hdl);

if (result != TDRV003_OK)
{
     /* handle error */
}
```

RETURNS

On success, TDRV003_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV003_ERR_INVALID_HANDLE	The specified TDRV003_HANDLE is invalid.

3.2.11 tdrv003WatchdogDisable

NAME

tdrv003WatchdogDisable – disable output watchdog

SYNOPSIS

```
TDRV003_STATUS tdrv003WatchdogDisable
(
    TDRV003_HANDLE          hdl
);
```

DESCRIPTION

This function disables the watchdog timer for the output lines.

PARAMETERS

hdl

This argument specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

EXAMPLE

```
#include “tdrv003api.h”

TDRV003_HANDLE   hdl;
TDRV003_STATUS   result;


/* Disable output watchdog */
result = tdrv003WatchdogDisable(hdl);

if (result != TDRV003_OK)
{
    /* handle error */
}
```

RETURNS

On success, TDRV003_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV003_ERR_INVALID_HANDLE	The specified TDRV003_HANDLE is invalid.

3.2.12 tdrv003WatchdogReset

NAME

tdrv003WatchdogReset – reset output watchdog error

SYNOPSIS

```
TDRV003_STATUS tdrv003WatchdogReset
(
     TDRV003_HANDLE              hdl
);
```

DESCRIPTION

This device function resets an output watchdog error. This function must be called after a device function returns the error code *TDRV003_ERR_IO.*

PARAMETERS

hdl

This argument specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

EXAMPLE

```
#include “tdrv003api.h”

TDRV003_HANDLE   hdl;
TDRV003_STATUS   result;


/* Reset watchdog error */
result = tdrv003WatchdogReset (hdl);

if (result != TDRV003_OK)
{
     /* handle error */
}
```

RETURNS

On success, TDRV003_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV003_ERR_INVALID_HANDLE	The specified TDRV003_HANDLE is invalid.

4 Legacy I/O System Functions

This chapter describes functions which are relevant only for the legacy TDRV003 driver.

4.1 tdrv003PciInit

NAME

tdrv003PciInit() – Generic PCI device initialization

SYNOPSIS

void tdrv003PciInit()

DESCRIPTION

This function is required only for Intel x86 VxWorks platforms. The purpose is to setup the MMU mapping for all required TDRV003 PCI spaces (base address register) and to enable the TDRV003 device for access.

The global variable *tdrv003Status* obtains the result of the device initialization and can be polled later by the application before the driver will be installed.

Value	Meaning
> 0	Initialization successfully completed. The value of tdrv003Status is equal to the number of mapped PCI spaces
0	No TDRV003 device found
< 0	Initialization failed. The value of (tdrv003Status & 0xFF) is equal to the number of mapped spaces until the error occurs. Possible cause: Too few entries for dynamic mappings in sysPhysMemDesc[]. Remedy: Add dummy entries as necessary (syslib.c).

EXAMPLE

```
extern void tdrv003Init();

…

tdrv003PciInit();

…
```

5 Debugging and Diagnostic

The TDRV003 device driver provides a function, and debug statements to display versatile information of the driver installation and status on the debugging console.

If the VxBus driver is used, the TDRV003 show routine is included in the driver by default and can be called from the VxWorks shell. If this function is not needed or program space is rare the function can be removed from the code by un-defining the macro INCLUDE_TDRV003_SHOW in tdrv003drv.c

The tdrv003Show function (only if VxBus is used) displays detailed information about probed modules, assignment of devices respective device names to probed TDRV003 modules and device statistics.

If TDRV003 modules were probed but no devices were created it may be helpful to enable debugging code inside the driver code by defining the macro TDRV003_DEBUG in tdrv003drv.c.

In contrast to VxBus TDRV003 devices, legacy TDRV003 devices must be created "manually". This will be done with the first call to the tdrv003Open API function.

```
-> tdrv003Show
Probed Modules:
    [0] TPMC671: Bus=4, Dev=2, DevId=0x029f, VenId=0x1498, Init=OK, vxDev=0xffff800000005580

Associated Devices:
    [0] TPMC671: /tdrv003/0

Device Statistics:
    /tdrv003/0:
        open count           = 0
        interrupt count      = 0
        Current Input Value  = 0x0000
        Current Output Value = 0x0000
        Active pending Jobs  = 0
        Debouncer disabled
        Watchdog disabled
```